7-2020

Cyber Space Odyssey: A Competitive, Team-Oriented Serious Game in Computer Networking

Kendra Graham

James Anderson

Conrad Rife

Pranav R. Patel

Scott L. Nykl
Air Force Institute of Technology

See next page for additional authors

Recommended Citation

K. Graham et al., "Cyberspace Odyssey: A Competitive Team-Oriented Serious Game in Computer Networking," in IEEE Transactions on Learning Technologies, vol. 13, no. 3, pp. 502-515, 1 July-Sept. 2020, doi: 10.1109/TLT.2020.3008607.

Cyberspace Odyssey: A Competitive Team-Oriented Serious Game in Computer Networking

Kendra Graham, James Anderson, Conrad Rife, Bryce Heitmeyer, Pranav R. Patel, Scott Nykl, Alan C. Lin, and Laurence D. Merkle

***Abstract*—Cyberspace odyssey (CSO) is a novel serious game supporting computer networking education by engaging students in a race to successfully perform various cybersecurity tasks in order to collect clues and solve a puzzle in virtual near-Earth three-dimensional space. Each team interacts with the game server through a dedicated client presenting a multimodal interface, using a game controller for navigation and various desktop computer networking *tools of the trade* for cybersecurity tasks on the game's physical network. Specifically, teams connect to wireless access points, use packet monitors to intercept network traffic, decrypt and reverse engineer that traffic, craft well-formed and meaningful responses, and transmit those responses. Successful completion of these physical network actions to solve a sequence of increasingly complex problems is necessary to progress through the virtual story-driven adventure. Use of the networking tools reinforces networking theory and offers hands-on practical training requisite for today's cyber operators. This article presents the learning outcomes targeted by a classroom intervention based on CSO, the design and implementation of the game, a pedagogical overview of the overall intervention, and four years of quantitative and qualitative data assessing its effectiveness.**

***Index Terms*—Computer networking education, cyber education, serious games.**

I. Introduction

TRADITIONAL learning formats, such as textbooks and lectures, can organize and present material, but long-term retention is improved when students internalize and apply concepts. It is widely accepted that education can benefit from more engaging instructional formats that facilitate motivation and interaction and allow students to apply and practice what they learn in controlled settings [1]–[3].

Dörner *et al.* [4] outline how the power of games extends beyond pure entertainment. In contrast to play, which does not necessarily have a purpose or explicit rules [5], games are rule based and have specific goals or win/lose conditions. This aspect of games enables them to serve clear educational purposes: students can receive feedback on their progress toward the learning objectives through a game's reward system. Furthermore, in contrast to traditional homework problems, game design focuses on providing pleasant sensory experiences, i.e., fun. This generates interest and curiosity that translates into motivation and active engagement, encouraging sustained learning. In particular, Graesser and Ottati [6] found that interactivity and narratives within games promote sustained learning. In turn, the choices made during a game uniquely create an emotional attachment that promotes quicker comprehension and better retention. In essence, there are few equivalent education tools that convey a "spirit of play instead of work" [7].

Cyberspace odyssey (CSO) is a serious game (i.e., a game in which the primary purpose is something other than entertainment [8]) designed to exploit this opportunity, specifically for the topic of computer network security. It allows students to learn and practice computer network basics in the context of a networked, multiplayer game that integrates real-world components with an engaging three-dimensional (3-D) virtual environment. Its design is heavily influenced by the target audience and the learning outcomes (LOs). Other key design features focus on encouraging competitiveness and fun play [4], [9].

The following section places CSO within the context of relevant prior research. This article then describes the game in relation to its audience and the relevant LOs of the academic program of which it is a part. It next discusses the system design and implementation, again in relation to the relevant LOs. This article continues by describing the overall intervention (pregame briefing, reference materials, pregame orientation period, and gameplay) and the assessment of its impact on student learning. The assessment includes analysis of pretest and posttest responses, a student survey at the end of the course, instructor observations, and quantitative score and game objective completion rates.

This article concludes by identifying game enhancements to be included for future course offerings as part of a continuous

Manuscript received July 27, 2018; revised January 25, 2019, August 23, 2019, and March 26, 2020; accepted July 3, 2020. Date of publication July 10, 2020; date of current version September 16, 2020. *(Corresponding author: Scott Nykl.)*

Kendra Graham was with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433 USA. She is now with Brigham Young University, Provo, UT 84602 USA (e-mail: deloris4@gmail.com).

James Anderson and Bryce Heitmeyer were with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433 USA. They are now with Wright State University, Dayton, OH 45435 USA (e-mail: anderson.10@wright.edu; heitmeyer.9@wright.edu).

Conrad Rife was with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433 USA. He is now with Brigham Young University—Idaho, Rexburg, ID 83460 USA (e-mail: rife.conrad@gmail.com).

Pranav R. Patel, Scott Nykl, Alan C. Lin, and Laurence D. Merkle are with the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433 USA (e-mail: prpatel@ieee.org; scott.nykl@afit.edu; alan.lin@afit.edu; Laurence.Merkle@afit.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TLT.2020.3008607

improvement process. (The supplementary pages present the pre/posttest instrument.)

II. Related Work

A substantial and growing body of research explores the ability of games to teach computer science concepts in an engaging and effective way (see [7] for a recent review). However, a relatively small fraction of serious games intended for computing education focus on cybersecurity, and only a handful of such games are aimed at information technology (IT) professionals [10]. Notable examples include CyberCIEGE, CounterMeasures, SecurityCom, CyberProtect, and HackNet.

CyberCIEGE [11] was developed for large-scale information security training and awareness purposes by the Naval Postgraduate School (NPS) and is available at no cost to government organizations and educational institutions. It is a single-player graphical user interface (GUI)-based resource management simulation game that places the user in charge of information assurance decisions for an enterprise environment. Players attempt to protect information against attacks while still making it accessible to those who require access.

The simulation is executed by an extensible game engine, which provides functionality for developers to create different scenarios for varied training purposes. The basic user scenario is intended to educate information system users with respect to fundamental computer security topics, including information value, access control mechanisms, social engineering, password management, malicious software, basic safe computing, safeguarding data, and physical security mechanisms. The overall goal is to improve their everyday cyber decision-making processes. The second scenario developed by the NPS is intended for IT staff and introduces them to "physical security mechanisms, access control, filtering, antivirus protection, data backups, patching configurations, password policies, and network vulnerability assessment." Scenarios developed by other researchers focus on identity theft prevention and reducing the risks of distributing worms and viruses. Chang and Chua later constructed a scenario to teach network traffic analysis in the context of simulated transmission control protocol (TCP) Internet protocol (IP) synchronize flood and packet sniffing attacks using a custom analysis tool integrated directly into the CyberCIEGE interface [12]. None of these scenarios employ real-world network traffic, use ubiquitous traffic analysis tools, such as Wireshark [13], or require network packet construction.

CounterMeasures [14], also a single-player simulation game, styles itself as a more engaging and effective alternative to traditional textbooks for learning security principles. Players are instructed in basic hacking techniques as they are given objectives to accomplish in a sandboxed shell environment. The game is implemented with a client–server architecture, in which the client consists of a simple GUI, including a console window. Players send commands through the GUI to the server, which uses one virtual machine to execute those commands, targeting the other virtual machine. The server then sends the appropriate responses and helpful instructions to the client for display in the GUI.

SecurityCom [15], which has similar training objectives to CyberCIEGE, is simpler and designed for team-based information security education. Gameplay in SecurityCom requires players to negotiate and work with each other to make information assurance decisions. Although players work in teams, there does not seem to be support for multiple teams.

CyberProtect [16], sponsored by the Department of Defense, was a 2010 finalist in the Serious Games Showcase and Challenge (SGS&C). To qualify as a serious game for SGS&C, a game must clearly state measurable learning objectives, provide players with a clearly identified problems, make use of gameplay dynamics, and provide player feedback regarding progress toward learning objectives [17]. CyberProtect puts the player in the role of a network administrator for the purpose of learning basic computer network defense and information assurance resource management through a quarterly budget mechanism. The game is still available at no charge and is the only entry currently listed under Defense Information Systems Agency Online Training Catalog for Cybersecurity Simulations.

HackNet [18] was both a 2015 SGS&C finalist and a winner of the 2015 Simulation Technology and Training Conference held annually by Simulation Australasia. It lets the player assume the role of an offensive hacker, using UNIX-like commands and metasploit-like tools. The narrative encourages ethical white-hat typical behavior, stepping the player through progressively harder challenges that combine lessons learned from previous missions.

The animations developed by Yuan *et al.* [19] to teach network security concepts do not meet the definition of games, but have enough game-like elements and teach concepts similar enough to those taught by CSO to be relevant. These tools show visualizations of several computer security concepts related to networks, such as a packet sniffer, the Kerberos authentication architecture, and demonstrations of wireless network attacks. Each visual has several elements that allow the user to interact with it, in ways such as entering their own data input or controlling the flow of the animation by skipping forward and backward, pausing, or speeding up and slowing down the animation.

CSO distinguishes itself from these games in the way that it integrates a virtual game world with a real-world networking apparatus. CyberCIEGE does provide a virtual world and motivating story, but the game is purely virtual and does not teach the same networking concepts that CSO does. SecurityCom is similar to CyberCIEGE, but is simpler and designed for teams rather than individuals. The visualization tools teach some of the same things that CSO does, but do not provide practice for the student. CounterMeasures could likely be configured to teach the same principles as CSO, but the practice would still come by way of a simulation, and each practice exercise would still be an end unto itself, without an overarching motivational premise. CSO provides real-world practice in packet capture and crafting, combined with a 3-D virtual world to give context and motivation for the practice. Furthermore, CSO is multiplayer rather than single-player, requiring team members to cooperate with each other while encouraging competition between teams and thereby making the game nondeterministic.

TABLE I
CSO FORMAL LOS AND THEIR CORRESPONDENCE TO CS2013 LOS

CSO Outcome	*CS2013 Outcome(s)*
Recognize and differentiate between the layers of the TCP/IP protocol suite (as defined by Kurose [23]; see Table II for the specific implementations of the layers used in the CSO activity)	NC/Introduction "Describe the layered structure of a typical networked architecture." [Familiarity]
Given a specific TCP/IP layer, create and use a valid data payload for that layer	NC/Networked Applications "Implement a simple client-server socket-based application." [Usage]
Capture traffic and decode data with Wireshark	IAS/Digital Forensics, "Capture and interpret network traffic." [Usage]
Synthesize and transmit semantically valid UDP packets	NC/Networked Applications "Implement a simple client-server socket-based application." [Usage]

CSO also distinguishes itself from many other serious games aimed at facilitating computing concepts by avoiding several pitfalls identified by the Innovation and Technology in Computer Science Education 2016 working group [7].

1) CSO's LOs are explicitly identified.
2) These LOs are in the networking domain rather than an introductory computer science domain.
3) CSO has been in practical use for four years, and it is anticipated to be for the foreseeable future.
4) CSO is one part of an intervention, the efficacy of which in facilitating its LOs has been evaluated each year, and those evaluations are used to enable continued improvements.

III. CONTEXT OF THE CSO GAME AND INTERVENTION

A. Understand the Audience

CSO is used in the Advanced Cyber Education (ACE) program, a four-week leadership training course teaching cybersecurity principles. ACE is offered each summer by the Center for Cyberspace Research at the Air Force Institute of Technology (AFIT). It is open to selected cadets of the Reserve Officer Training Corps (ROTC) programs and service academies of the United States. They must meet a minimum current college grade point average requirement, receive a recommendation from their commanding officer, and have significant background knowledge in computer science. The majority are drawn from Air Force ROTC and the U.S. Air Force Academy, with a small number of Army ROTC and U.S. Military Academy (West Point) cadets. Thus, they are undergraduates and primarily male, between the ages of 18 and 24, and majoring in technical disciplines. The military training in their backgrounds instills a focus on teamwork, which is explicitly identified as a part of the "Excellence in All We Do" Air Force Core Value [20]. They are also familiar with the Department of Defense Cyber Strategy [21], which calls for the use of team competitions to "identify the most capable DoD military and civilian cyber specialists." Finally, the cadets receive evaluations of their ACE performance, which are used as discriminators in their military performance reports. Thus, they are motivated to perform well.

TABLE II
CSO IMPLEMENTATION OF TCP/IP SUITE LAYERS

TCP/IP Layer	*Implementation*
5: Application	3D Virtual World (see Sec. V)
4: Transport	UDP
3: Network	IP
2: Link	802.11n
1: Physical	2.4 GHz Radio Waves

TABLE III
BLOOM'S TAXONOMY OF CSO LEARNING ACTIVITIES
(SEE SUPPLEMENTARY MATERIAL FOR COMPLETE TEST)

Level (Bloom)	*Test Items*	*Activities*
Knowledge	1–2, 4–11	Pre-Game Briefing Navigate Spacecraft
Comprehension	3, 12–13, 18–22	Hail Space Stations
Application	14, 16–17	Capture Packets (Wireshark) Transmit Packets (PacketCrafter)
Analysis	15	Decode Packets Decode Cyphers Deduce Suspect
Synthesis		Craft Packets Semantically Interpret Responses Submit Accusation
Evaluation		Decide Between Alternative Strategies

B. Learning Outcomes

The ACE program covers a broad range of cyber concepts. Many of the program's formal LOs coincide with outcomes found in the ACM/IEEE Computer Science Curriculum 2013 (CS2013) report [22]. Some of those ACE outcomes with which previous program participants have struggled have been adopted as outcomes for the CSO intervention. They are listed in Table I, which also relates them to their corresponding CS2013 knowledge areas, LOs, and levels of mastery. The CS2013 Information Assurance and Security (IAS) knowledge area and the Networking and Communication (NC) knowledge area pertain most relevantly to CSO. CS2013 defines the Familiarity level of mastery as "the student understands what a concept is or what it means," and the Usage level as "the student is able to use or apply a concept in a concrete way."

They are specific enough to be evaluated quantitatively using a pre/posttest and qualitatively using a participant questionnaire.

The CS2013 Familiarity level of mastery corresponds closely to the knowledge and comprehension levels of the cognitive domain of Bloom's taxonomy [24], while the CS2013 Usage level corresponds closely to Bloom's application and analysis levels. In order to successfully complete the game, CSO requires participants to complete a variety of tasks relating to the LOs, sequenced to require learning at successively higher levels of Bloom's taxonomy (see Table III). These tasks also span the range of network abstraction layers: connect to wireless access points, use packet monitors to read blocks of raw bytes from a network stream, decrypt network messages, reverse engineer network protocols, and craft and transmit well-formed packets containing semantically relevant data to exploit the aforementioned network protocols.

Furthermore, CSO includes activities that go beyond the formal outcomes to at least Bloom's synthesis level, and arguably to the evaluation level, as well as activities directed toward the affective domain such as fostering teamwork and establishing a collaborative learning environment. Finally, the specific tools chosen for use in CSO have widespread real-world application. All text with which students interact is represented using the American Standard Code for Information Interchange (ASCII) code, which together with its superset Unicode is employed by essentially all real-world applications. All network protocols used in the game are based on the user datagram protocol (UDP), which underpins all real-world streaming services (audio, video, etc.). The primary software tools available to the students (e.g., Wireshark, Boost C++ libraries) are commonly employed as real-world packet inspection and construction tools.

C. Decide Game Usage

The first week of the ACE program presents a general introduction to computer networking, followed by an overview of cyber attack and network defense. These topics help set the general cyber-landscape in which the students are immersed, but do not directly cover the LOs of the CSO intervention. CSO is played during the second week of the course. The intervention is monitored by two instructors and consists of a preparation phase (pretest and pregame briefing/lecture), execution phase (gameplay), and after-action review phase (debriefing/discussion and posttest) [8]. In years 2016–2018, the pretest was given *before* the briefing/lecture. This potentially leads to ambiguity in what facilitated learning. In 2019, this was changed to first give the briefing/lecture. Subsequently, we administered the pretest, followed by gameplay, and concluded with the posttest. In this latter scenario, the only activity between the pre- and posttest was CSO itself—removing the learning ambiguity between the briefing/lecture and the game. The statistics presented in Section VI-A show that learning still occurred in this revised paradigm.

D. Design in Fun

The action of the game takes place in both the virtual and real worlds. In the former, teams must navigate a spacecraft (a.k.a. "ship") to certain locations, while in the latter, they must use real-world programming tools to perform network operations. Success in the game hinges on coordinating actions in the two domains, ensuring that while participants learn from the game activities, they experience them as means toward the end of completing game objectives.

IV. Game

A. Overview

CSO is a puzzle-solving game in which each team uses computer and network "hacking" skills as well as critical thinking to solve a logical puzzle. Its interface is multimodal [25], with players using a game controller to navigate the 3-D environment as well as a variety of desktop applications and environments to perform other game actions. A YouTube video providing more detailed description is available [26].

B. Mechanics, Dynamics, and Aesthetics

As a game, CSO can be understood through the Mechanics, Dynamics, and Aesthetics (MDA) model for game design [27]. Mechanics comprise the rules and bounds of player actions. Dynamics are the elements of game flow, which can be viewed as the player (or team) behavior emerging from the combination of their goals and the actions possible through the mechanics. Finally, aesthetics are the experiential elements resulting during and following gameplay.

Within this framework, C++ programming, network packet capture, and other hacking skills are examples of mechanics, as are game controller inputs, since they are among the fundamental actions available within the game. The dynamics of CSO are extensive. For example, the scoring system influences game flow by rewarding the use of mechanics in a sequence that leads to identification of the spy. In particular, repetitive behaviors yield diminishing rewards, thereby discouraging teams from delaying progress. Overall, the dynamics of the scoring system result in a competitive aesthetic by motivating teams to solve the problem faster than their competition.

Overall, CSO's aesthetics encourage teams to get through as many stages of the game as possible. In turn, the dynamics are such that in order to continue making progress, teams must continue mastering new hacking skill mechanics, each of which maps to one or more LOs. The game's aesthetics also inherently support the affective domain outcomes. Teams that work well together and understand each member's role will progress faster than other teams. Independent of winning, players belonging to such teams tend to enjoy the overall experience more than those on teams that do not communicate effectively; thereby, reinforcing the affective domain outcomes.

A concrete example is illustrative. Once teams have mastered the hacking skills required to craft network packets, which occurs early in the game, they have the ability to "hack" other teams by injecting malicious control packets (a mechanic of the game) that disrupt another team's ability to progress. However, continually doing so is intentionally designed to be a failing strategy. Specifically, teams receive a modest number of points for their first hack and decreasing rewards thereafter. This results in the dynamic that it is only advantageous to hack the lead team, and to do so only a few times at most. This, in turn, results in the aesthetic of a race, in which each team is primarily focused on completing the puzzle rather than interfering with other teams.

Scott [28] extended the MDA model to include the player culture, which may influence both the dynamics and the aesthetics. Since the audience of CSO has been military cadets, who are acclimated and receptive to friendly competitive settings, the scoring system is familiar. Cadets are also trained in a culture that promotes teamwork, which is consistent with the intended aesthetic design of the game.

C. Gameplay

The premise of CSO is that the host institution (specified in the configuration file, and hereafter assumed to be AFIT) has received intelligence that a terrorist group seeks to infiltrate the organization. A key element of the group's plan is the

Fig. 1. Game World. This view includes Earth, the communication ranges ("bubbles") of the five space stations (colored to indicate the team to which they belong), and the background. Teams would be unlikely to be this far from the space stations during an actual game.

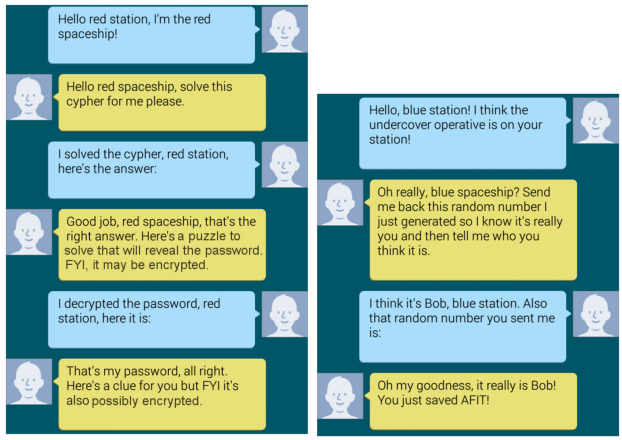
Fig. 2. Client view of the Game World during gameplay. This view shows the "dashboard" of the client's ship, the Pacific Ocean side of the Earth, the red ship, a space station within communication range, a distant space station's communication bubble, the scores, the distances to the other ships, the console, and the background.

hijacking of a space station and use of its communication capabilities to breach AFIT's security. A clandestine agent working at one of the five space stations (a.k.a. the "spy," the "culprit") will execute that portion of the plan. To thwart the plan, AFIT has engaged the services of five bounty hunter teams to identify the spy (a.k.a. "solve the mystery") and collect enough evidence (a.k.a. "clues") for conviction.

The game is played in a 3-D virtual world (see Figs. 1 and 2) that includes the Earth, as well as one space station and one spacecraft for each team. Each spacecraft is initially docked at the team's space station. These elements are displayed within each team's minimal GUI, which also shows their current progress in the game, their scores and those of the other teams, and their ship's distance from those of the other teams. In addition, transparent console-like text displays appear occasionally to provide contextually relevant information.

Clues are gathered by inspecting the space stations' (simulated) "private networks," as illustrated in Fig. 3(a).

More specifically (all messages use the UDP), we have the following.



(a) Clue Collecting (b) Making an Accusation

Fig. 3. These images were used to explain the mystery solving process to the cadets playing the game. (a) Process of collecting a clue. (b) Process of making an accusation.

1) The team maneuvers their spacecraft within communications range of a space station and sends a "hail" message to the server in the form of a packet containing their team identifier (ID) and the ID of the station they are hailing.
2) The server acknowledges the hail with its own packet containing an arithmetic problem in the form of ASCII text encrypted with a simple cipher using a secret key.
3) The team eavesdrops over the local wireless network to capture and identify the server's acknowledgment, determines the secret key, decrypts the ASCII text, and solves the arithmetic problem to obtain the "station key." They then craft a packet containing the station key, the team ID, and the station ID. They send this packet to the server.
4) The server again responds with its own packet, this time containing the "station password" in the form of ASCII text encrypted with a simple Caesar Cipher using the station key.
5) The team finds the server's response, decrypts the station password, crafts a packet containing the password along with the team and station IDs, and sends it to the server.
6) The server responds with a packet containing a clue encoded in alphabetic ASCII text with a keyword substitution cipher using the station password as the key.
7) The team finds the server's response and decrypts the clue.

The clues form a simple logic puzzle, designed such that each clue constrains the list of potential suspects and all clues are necessary to constrain the list to a single suspect. Once a team knows who the culprit is, they follow a two-step authentication protocol in order to make a formal accusation [see Fig. 3(b)]. First, the team maneuvers to the culprit's space station and sends a request to initiate the accusation process. The server responds by sending a single-use randomly generated authentication code. Finally, the team returns the code to the server along with the name of the culprit. If they have obtained

all of the clues and their accusation is correct, then the team completes the game objectives. If they somehow managed to make a correct accusation without having all the clues, they have identified but failed to convict the culprit. In this case, they must go back and collect the missing clues in order to have enough evidence for an actual conviction to complete game objectives.

Finally, as mentioned previously, teams have the ability to impede each other's progress by "hacking" each other. Specifically, a hack causes the target to lose control of one of their spacecraft's degrees of freedom (roll, pitch, yaw, or forward/backward translation) for a certain length of time. This means that the spacecraft will either fly backwards or spin around one of its rotational axes until the hack times out. A team originates a hack against a victim team by navigating within range, crafting a packet specifying the hacking team and the victim team, and transmitting this packet to the server. The server verifies that the hacking and victim teams are within range and then implements the hack and awards points to the hacking team. However, the server does not verify that the hack was originated by the hacking team identified in the packet. Thus, teams can legally send hacks posing as another team, thereby circumventing the time limit. However, doing so awards points to the impersonated team.

D. Increasing Competition

The main objective of CSO is to find and convict the culprit, but because it is a multiplayer game (with each player being a team of approximately three to eight people in this case), several elements are included specifically to increase competition. First, the story itself sets the stage by casting the players as bounty hunters—competition with others is an inherent aspect of any bounty hunter's job.

The scoring system is also designed to increase competition. Score is the tiebreaker among teams that complete the mission, as well as among those that do not. Points are awarded primarily for progress toward game objectives, with teams that complete objectives earlier earning more points. For example, the first team to hail a particular space station earns 15 points, while the second earns 14, the third 13, and so on. Although a single point has a small effect on the overall outcome of the game, because all objectives are scored similarly, large point differences between teams can accumulate over the course of the game. This is the case even when teams eventually complete the same set of objectives.

Finally, in order to prevent teams from overusing the "hacking" feature, in addition to the scoring mechanism mentioned previously, once a team sends a hack, they lose the ability to do so for the same length of time as the hack's duration.

E. Team Organization

The process of obtaining clues is complex enough to justify distributing team responsibilities; therefore, teams are strongly encouraged to assign each member to one of four roles. The pilot is responsible for navigating the spacecraft through the virtual world. As such, they are in the best position to maintain awareness of the team's situation, guide its strategy, and coordinate their teammates' activities. Team members performing the second role ("packet capturers") use Wireshark to capture network traffic and identify those packets that are relevant to the team's current goals. The third role is the cryptographer. Team members in this role decrypt the information obtained from the captured packets through correct usage of the provided decryption utilities and then interpret the decrypted information to their teams. The last role is to customize provided C++ software to craft and send network packets that affect the team's progress in the game.

With fewer than four team members, players can combine roles. With more than four, assigning multiple team members to capture and craft packets is most effective. Regardless, in order to succeed, team members must use all of the skills mentioned in the LOs of the game, especially eavesdropping over a network to find packets and constructing their own packets to send. They must also coordinate with each other effectively: the pilot must be in the right geospatial position during the entire exchange to maintain a comm link, the packet capturers need to know when a packet is sent to/from the server so they can be listening to the network, someone must decrypt the information received to semantically process the information before packets can be sent, and the packet crafters need the information obtained from the packet sniffers to craft the next message in the sequence.

V. Implementation

A. Game Engine

CSO is powered by the AFTRBurner game engine [29], a project created at Ohio University and funded by the National Science Foundation Graduate K-12 program. AFTRBurner was designed to provide a powerful yet flexible set of tools to create high-quality educational games that take advantage of an immersive 3-D virtual world environment. The 3-D world supported by the game engine includes Open-Graphics-Library-based graphics, built-in animation capability, an advanced physics engine, and support for event-driven gameplay. In addition, the AFTRBurner engine has its own client–server networking architecture allowing gameplay over a network. The engine was written in platform-independent C++ to allow for cross-platform use, and its object-oriented design facilitates extensibility and ease of use for developers.

B. Code Structure

Like the engine, the CSO game itself is written in platform-independent object-oriented C++ (see Fig. 4). CMake is used for cross-platform build configurations, and the compiled game can easily be configured to run as either a server or a client.

The majority of the classes used in programming the game inherit from engine classes. The game's top-level manager class holds graphics and data objects, sends and receives network messages, collects and processes hardware input, and renders the graphics.

Most graphics objects used are default objects included with the engine, but new classes for the planet and the spacecraft

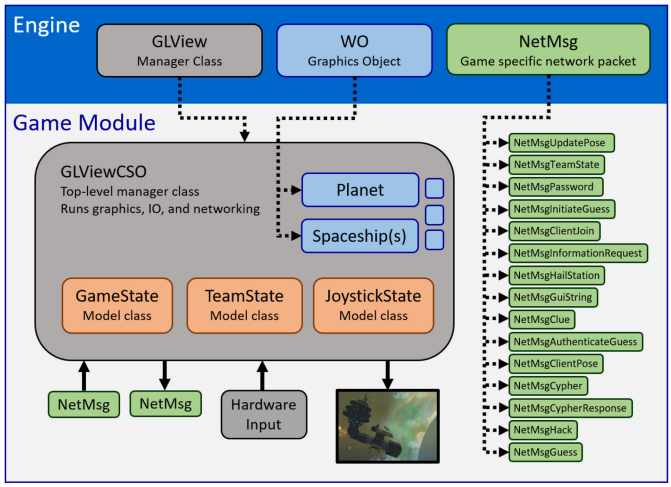


Fig. 4. Diagram showing code structure. Dotted lines show inheritance. Solid arrows indicate input or output.

have been defined in order to give them custom behavior. In particular, the spacecraft objects are responsible for correctly updating their own positions when called upon to do so by the manager class. Among the default objects used were two-dimensional (2-D) text boxes that formed a dynamic on-screen console-like display to show game messages. All warnings and informational messages for the players were printed using the on-screen console.

C. Network Components

Game state data are managed using a client–server architecture, with copies of the same executable acting as server and client depending on their configuration files. Each team is represented as one client represented as a spacecraft that authoritatively owns its pose data (position and orientation). The server owns the authoritative copy of all other game state data, sending elements of it to the clients as necessary. For example, the server simulates the internal networks of the space stations. Because the clients and server have identically structured versions of all game state data, maintaining the game state requires only minimal network traffic.

The bulk of that traffic consists of spacecraft pose data, which is generated as follows. Event handlers in each client process joystick input values to calculate the associated ship's translational and angular velocity, yielding the ship's pose (position and orientation). Sixty times per second, each client transmits its pose to the server. With the same frequency, the server broadcasts all ship pose data to the clients, which update their copies of the other ships' poses.

In addition to its pose, the "Team State" for each spacecraft includes a Boolean variable for each degree of freedom, owned by the server. When one of these is true, the joystick event handler overrides the input for the associated degree of freedom and instead uses a predetermined constant as the corresponding velocity component. For example, if the Team State variable controlling forward thrust is set to true, the spacecraft moves backward at a constant velocity regardless of the joystick input. Once per second, the server broadcasts the Team State data, as well as all team scores and additional Boolean values describing the teams' progress. Clients then update their copies of their own Team State Boolean values, as well as their leaderboards and their on-screen summaries of team progress.

The various updates just described are implemented using the Net Message class provided by the game engine. In fact, all game messages inherit from the Net Message class. The content of a Net Message is contained entirely within the data payload of a typical network packet, and the engine automatically wraps it with the appropriate headers when it is sent. As such, a Net Message is reusable in the sense that the engine can wrap it and send it to one destination (IP address and port), then wrap it again and send it to a second destination. In CSO, all packets are UDP datagrams.

The Net Message format is simple and, in particular, includes an integer header ID in the data payload specifying the type of game message. This enables the game engine to polymorphically send messages and process those it receives. In addition, the mapping of Net Message subclass to header ID is the only information required for a third-party (i.e., neither a client nor the server) to both interpret captured game packets and craft valid packets. Conveniently, for the players, the mapping is displayed in the console window of every running instance of CSO. Once they discover this, they are able to gain additional awareness of the game state by capturing and interpreting packets, as well as to achieve a tremendous variety of effects (e.g., spoofing a spacecraft's location) by crafting and sending their own packets.

The ability for third parties to intervene in the game's internal network is an essential aspect of CSO. It is the mechanism by which the game integrates the real and virtual worlds, thereby relating directly to its LOs. As such, other than maneuvering spacecraft and hailing space stations, progress is impossible without learning how to capture, interpret, craft, and send packets. First, each piece of information needed to proceed to the next step can be obtained only by capturing and interpreting network packets. For example, when the server responds to a hail on behalf of a station, the client's GUI displays a message that the station sent a cipher, but does not display the cipher. Similar statements apply to passwords and clues. Next, progress at the more advanced stages of the game requires teams to identify packets sent to IP addresses other than their own, since the server sends station passwords and clues to the client color corresponding to the station's color. Furthermore, teams must learn to identify packets based on their transmission times, since each authentication key is invalidated when another one is requested by the same client. Finally, packets containing the answer to a station cipher, the decrypted station password, the beginning of an accusation, and the final accusation must all be crafted and sent manually.

Despite its complexity, a few features keep the game difficulty from rising to the level of discouraging players. One such feature is the appearance of the message header ID in clear ASCII text in the data payload of almost every packet that teams are meant to find. This greatly facilitates the teams' ability to find these packets. The one exception is the

accusation authentication key. By that point in the game, teams have extensive practice finding packets and should have no trouble finding it based on other criteria. Another feature mitigating the game difficulty is the frequent on-screen feedback that teams receive. They are notified whenever they send a valid packet to the server, as well as what went wrong if their packet was not valid. For example, if one team tries to hack another when their ships are too far apart, the server sends a message that the hack was not accepted because they were out of range. The on-screen feedback is initiated by the server, which sends Net Messages that trigger the client to print their payload strings to the console.

D. Robustness

Since the game requires teams to craft and send their own packets to the server, both the server and the logic of the game itself are designed to be robust against both malformed and malicious packets. Because the hardware and software tools provided to the players have been in widespread public use for considerable time, it is assumed that the game is secure at the physical, link, and network layers. Nonetheless, in case a failure occurs at one of these layers, game state is backed up to a disk file once per second. As such, either individual clients or the entire game can be restarted at any time with no adverse effects.

At the transport layer, all game packet types except the initial client connection request use the UDP. Because that protocol is connectionless (i.e., does not require the recipient to respond), it does not require blocking or buffering. In other words, no combination of packets can force the server's (or client's) transport layer into an unsafe state or overflow its buffer. The tradeoff is that it is theoretically possible for a packet to be dropped if the recipient is not ready to receive it. Because of the relatively low number of packets required by CSO, this has not been observed in practice. If it did, the only consequence would be that the recipient would not respond to the message at the application layer.

Robustness at the application layer derives mainly from the fact that syntactically incorrect messages are simply rejected, i.e., not passed to the game's message handlers. As such, those handlers need only verify the semantic correctness of the message contents, which is not complicated. For example, many messages specify the client claiming to be the sender, and the server must verify that such a sender exists within the game. This is trivial, since the client is specified using the index of the client, which must belong to a set of consecutive integers within a known range (typically zero to four). Other aspects of semantic correctness are verified with the same ease.

Assuming that a message is syntactically valid, its effect will be at the level of gameplay, and in accordance with the educational philosophy of CSO, it is, therefore, considered a valid game mechanic. Certainly, some such messages can be used to accomplish effects outside of the nominal flow of the game. In particular, because neither the server nor the clients verify that the identity of the sender claimed in a message matches the actual sender, it is easy for one team to spoof another. These "vulnerabilities" are intentional. The purpose of the game is for players to practice intercepting, interpreting, crafting, and sending network packets, and the opportunity to exploit these vulnerabilities encourages them to go beyond the bare minimum in developing these skills.

Finally, gameplay also involves the social layer. The authors have not observed any significant difficulties with the game at this level within the context of an instructor present to organize, explain, etc. We hypothesize that if one team was abusing the openness of the game mechanics to make it effectively impossible for other teams to play (e.g., by flooding the server with packets), we could just ask them to stop. The game would then resume normally.

E. Representative Creative Game Mechanics

On the most basic level, a creative player could cause C strings to appear on their own or other team's screen. However, teams can clear their screens of messages, so this would not have a lasting effect on the game. It could cause the most interference if the player sent a continuous stream of them, keeping the opposing team from seeing messages from the server, but the on-screen messages are not essential to gameplay, and if the opposing team really wanted to see the true messages from the server, they could find those messages in Wireshark.

The next simplest thing for teams to do would be to send normal game packets as though they were a different client. This is less likely to have an effect, since the server verifies a client's location every time it thinks it receives a message from that client. Therefore, if the Red team pretends to be the Yellow team in an attempt to hack the Blue team, it will fail unless the Yellow team happens to be near the Blue team. Similar checks are performed when a client hails a space station, responds to a station's cipher, or sends a station password. Even if the target team happens to be in the right location for what the player is trying to do, the effect would either be positive or neutral for the targeted team: if the Red team sent a hail message pretending to be the Yellow team, the Yellow team would get the points for it; furthermore, sending incorrect answers to a cipher or sending incorrect passwords does not cause a team to lose points.

The accusation process could be used to cause another team to lose points, but it would be more difficult. First, the target team would have to be within range of the culprit's space station, or an authentication code would not be sent. The attacker would then have to find the authentication code and send an incorrect accusation to the server quickly enough that the target team would not have time to request another authentication code, thereby invalidating the one found by the attacker. The team could also defend itself by flying out of range of the culprit's space station, which would cause the attacker's false accusation to be rejected. Furthermore, attackers could only do this before a team makes a correct accusation; once a team has made a correct accusation, no more accusation attempts from them are accepted by the server. An attacker could have a greater effect by sending so many requests for authentication

codes that the target player is unable to find one and send a guess before the code expires. The target team could still defend itself by maneuvering away or using its self-destruct button to return to its home station if it is unable to move.

An attacker could also send messages that are normally sent by the server to a client, with varying results. The easiest thing to do would be to craft fake ciphers, passwords, and clues to send to the client. These have the potential to confuse a client, but would be easy to identify, since the sender IP would not be the server's actual IP. Even if the attacker could spoof its own IP address, it cannot replace the correct information sent by the server, and it is still possible for the target team to keep playing without much trouble.

A more insidious attacker could broadcast a malicious version of the Team State that the server sends out once per second. This could cause teams to be permanently hacked as well as have incorrect information about their scores and their progress in the game. If a team did this, it would quickly become obvious what was going on, and there would be no lasting effects once the attacker ceased its activity. Regardless, it is unlikely that a team would figure out how to do this effectively. The documentation does not provide the information necessary to intentionally construct a malicious Team State, and players do not have access to the source code. The most difficult attack for a team to perform would be the transmission of a malicious version of the clients' pose messages; this would have an interesting but mostly harmless effect: players would not be able to accurately see other spacecraft, but their own positions would remain unaffected, since the clients themselves own the authoritative copies of their positions. Furthermore, this effect would only last as long as the attack and would not cause any permanent damage.

Theoretically, it is possible for clients to spoof spacecraft locations, either their own or that of another spacecraft, but realistically, it would be nearly impossible during gameplay to find valid coordinates to use for a spoofed location. Without valid coordinates for locations one wishes to spoof, location spoofing is useless. Even with valid coordinates, location spoofing is difficult to get right. If a team manages to figure it out during gameplay, they deserve whatever advantage location spoofing can give them. The same principle applies for the rest of the game; the difficulty involved in any creative use of network messages is high enough that any team that figures it out deserves whatever advantage it gives them.

F. Generating the Mystery

As discussed above, gameplay involves solving ciphers, decrypting passwords, decrypting clues, and using the clues to solve a mystery. Accordingly, the following information must be assigned for each station:

1) an arithmetic expression, the value of which serves as the key of a Caesar cipher;
2) an isogram (i.e., word without repeating letters) referred to as the "password" and serving as the key of a keyword substitution cipher;
3) a clue to the mystery.

TABLE IV
CLUE TEMPLATES

Clue 1	The terrorist cell formed in July 2014, so only employees hired after that date are suspect.
Clue 2	———'s space station is missing crucial hardware that the terrorist group would require in order to carry out their plan, so none of its employees are suspect.
Clue 3	These employees have the technical background necessary to hijack a space station: ———
Clue 4	Here is a list of all employees hired since July 2014: ———
Clue 5	——— and ——— work on the Red station; ——— and ——— work on the Yellow station; ——— and ——— work on the Green station; ——— and ——— work on the Blue station; ——— and ——— work on the Cyan station.

Early implementations used fixed sets of ciphers, passwords, and clues. However, this approach undermines the reusability of the game and makes it theoretically possible for players to cheat by obtaining access to the source code of the game. As such, all ciphers, passwords, and clues are now generated randomly, seeded by the current time, and encrypted by the server at initialization. The passwords are chosen at random without replacement from a large library of isograms. The methods to generate the arithmetic expressions and the clues are more complicated.

The value of each arithmetic expression is a distinct randomly selected integer $2 \leq a \leq 25$. For each expression, integers b and c are selected randomly and integers x and y obtained such that $a = (x \cdot y + b)/c$, the right-hand side of which is chosen to be the expression. Specifically, both b and c are randomly selected until $a \cdot c - b$ is composite (with $c \in \{2, 3\}$ if $a > 12$ and $4 \leq c \leq 12$ otherwise, as well as $1 \leq |b| \leq 10$). Then, x and y are chosen such that $x \cdot y = a \cdot c - b$. This process ensures that the arithmetic involved in evaluating the expression is human friendly.

Clues are formed by instantiating the templates shown in Table IV. Specifically, one space station is randomly selected to be designated as innocent and fill the blank in Clue 2. A list of 32 names is randomly permuted and allocated as evenly as possible to the stations with any leftover names being assigned to the innocent station. The culprit is selected randomly from the names assigned to the remaining stations. Observe that all names have the same unconditional probability of being selected, so players with previous gameplay experience do not have advance knowledge of the culprit's identity.

A list of individuals with the technical background to hijack a space station is constructed consisting of the following: 1) three names selected randomly from the innocent station; 2) the culprit; and 3) four others selected randomly from the remaining stations. After being randomly permuted, this list is used to fill the blank in Clue 3.

A list of recent hires is constructed consisting of the following: 1) two of the names previously selected from the innocent station; 2) the culprit; and 3) three new names selected randomly from the remaining stations. After being randomly permuted, this list is used to fill the blank in Clue 4.

The culprit and one other name selected randomly from the same station are used to fill the blanks for that station in Clue 5. The recent hires from the innocent station are used to fill that station's blanks. Two names selected randomly from each of the other stations are used to fill the remaining blanks.

Observe that the culprit is unique in being on both the list of individuals with the necessary technical background and the list of recent hires without being assigned to the innocent station. Consequently, the five clues constrain the candidate solution set of the logical puzzle to exactly one element. Furthermore, no proper subset of the clues does so.

The clues are randomly permuted and assigned to stations. The server encrypts the first two clues using the corresponding keyword substitution ciphers and the last one by layered Base64 encoding (a simple substitution cipher).

G. Experimental Freedom With the Physical Network

The networked structure of CSO requires the server to manage the state of the game. Since all significant game events consist of packets sent to the server, only the server knows when these events happen. Furthermore, clients do not know the addresses of other clients. Only the server knows the addresses of all the clients, so only the server is able to keep all of the clients updated with what they need to know. It does this by storing information about each team's progress in the game and updating it whenever a significant game event happens.

Because it always knows each team's progress, the server is able to back the data up to a file and reload it when necessary. This file's name depends on the time when the server started running and contains every team's progress in the game along with the data for the mystery generated. If the server crashes, it can be restarted and loaded with a backup file to resume the game from the point of the crash, overwriting the newly generated mystery information in favor of that which was stored in the backup file. A summary of game information is also written to a file once per second.

The server also has a sophisticated mechanism enabling clients to hack each other. They typically do directly, but as discussed above, the server maintains the true versions of the Team States and sends them to the clients once per second. As such, direct hacks are effective for no longer than one second, and the only way to affect another client's Team State for longer is to change the version sent by the server. This is achieved by sending a network packet to the server indicating the hacking team, the target team, and the type of hack. The server accepts a maximum of one hack from each team every 20 s, which is also the duration of each hack. The server continues to maintain the true versions of the Team States, but applies the active hacks to them before sending them to the clients. Clients cannot hack themselves.

Each ship's motion is handled by its client, based on joystick inputs. Specifically, ship velocity along each direction corresponding to a joystick axis is proportional to the square of a time-averaged value of the corresponding input. This allows a significant range of velocities while still providing precise control at low velocities. Thus, "normal physics" (e.g., Newton's Laws of Motion) are not used. Furthermore, there is no collision detection, either with other ships or with the space stations. However, spacecraft that navigate into the Earth or spend a significant time in Earth's atmosphere "die"—they are forcibly returned to their starting position and are unable to move for 30 s as a deterrent. Teams can also use a self-destruct button to achieve the same effect, which is useful in case they navigate so far from Earth that they cannot find their way back. Ships are warned when getting too close to or too far from the Earth and are prompted to hail a space station when they come in range.

H. External Elements

A number of external hardware and software components are necessary to implement the game, most of which operate independently of the game code. The most important of these are the networking hardware and software, C++ development environment, and GUI-based cryptography application.

Network communications occur over a local network. It is wireless so that an effectively unlimited number of clients (and associated players) can eavesdrop and inject packets. The network itself is unencrypted to make eavesdropping easier for the players. For the instances of the intervention offered to date, Windows workstations have been used as the client devices. Because implementation of monitor mode can be challenging with Windows network card drivers [13], separate machine configurations have been used for packet capture and injection. ALFA Universal-Serial-Bus-to-WiFi network cards have been used for those workstations that merely need to connect to the wireless network, and the Riverbed AirPcap adapter has been used in monitor mode to sniff packets. Wireshark has been used to filter and view packets captured via the AirPcap adapter. For packet injection, players have been provided with a template project written in C++ that provides several examples of how to send UDP datagrams, as well as an example of how to send the Net Messages used in the game.

Although it is merely a convenience for the players, a custom GUI-based cryptography application facilitates the Caesar Cipher and Keyword Substitution Cipher decryption required by the game. The application allows the user to enter ciphertext and a key, select an encryption type, decrypt the text, and see the resulting plaintext. It is implemented in C# with no external libraries and is, therefore, multiplatform.

VI. Assessment

In 2016 ($n = 39$), 2017 ($n = 41$), 2018 ($n = 43$), and 2019 ($n = 44$), ACE participants experienced the CSO intervention (pregame briefing, reference materials, pregame orientation period, and gameplay) as part of a 4-h laboratory. This section describes the assessment of the cadets' achievement of the formal LOs identified above by presenting the methodology, quantitative and qualitative observations, and interpretations thereof.

A. Methodology

The same instructor conducted the laboratory all four years. Cadets were first assigned random identification numbers to

enable correlation of their responses on the various instruments while still protecting their privacy. They were then given a pretest (see Supplementary Material) as well as a 30-min instructional presentation ("pregame briefing") comprising each team's objectives, how to navigate the virtual world, how to launch the requisite tools, and how to begin inspecting data packets sent across the wireless network. In the first three offerings, the pretest preceded the pregame briefing. In 2019, that order was reversed.

Next, the cadets were randomly assigned to one of five teams: Red, Yellow, Green, Blue, or Cyan. The cadets on each team negotiated amongst themselves to assign individual roles. Once the teams were formed, they received the game's required software, hardware, and documentation. The latter included a four-page reference guide summarizing the game story and mechanics (e.g., how to pilot the spacecraft, the processes of obtaining clues and making accusations), offering Wireshark filter suggestions, and specifying the network protocols and payload structures for the game's data packets. Both the pregame briefing and the reference guide consist of content at the knowledge and comprehension levels of Bloom's taxonomy. For example, the guide discusses which software options must be selected to attach to the wireless network and how to use the joysticks to navigate the spacecraft. Teams were then given 30 min to orient themselves (e.g., by reviewing documentation, exploring software, and learning to maneuver their spacecraft) followed by a single session of CSO lasting 2 h.

Cadets were not given access to the source code for the game itself.

During the game, five overhead projectors duplicated the teams' primary monitors, so that both instructors and other teams were able to observe their movements within the game environment, their scores, and their status messages. The instructor periodically switched between observing each of the five groups and was available for questions, but only interacted upon student request.

Intergroup communication was subjectively qualified—each team member used their own computer to gather information specific to their task. Without sharing this information or consuming additional information from their teammates, forward game progress would be impossible. Similarly, in this setup, no single student could perform all tasks necessary to "carry" her team by herself, because no single role has the ability to collect, process, and interpret all gameplay clues. Students were able to switch roles if they elected to do so, but there was no mandatory reorganization. The most common transition was from packet crafter to traffic interceptor or vice versa.

Following the game, the cadets completed a posttest (identical to the pretest). In each offering except 2018, they then completed a brief Likert-style questionnaire asking about their formal education in computing and their subjective assessment of the lesson. In 2016 and 2017, they responded to the following statements.

S1) "The content of the lesson was relevant."
S2) "The organization of this lesson was logical and easy to follow."

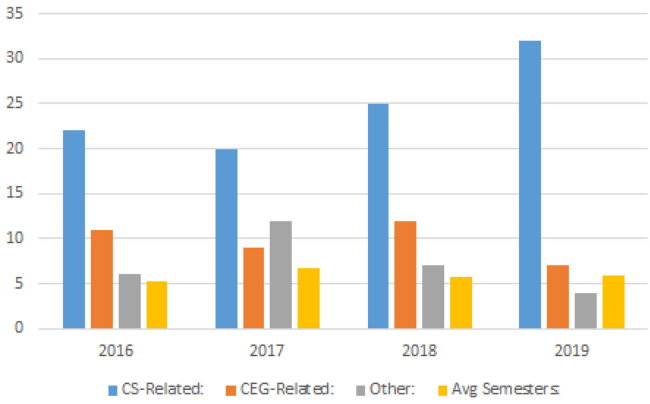


Fig. 5. Formal computing education by Year. Majors categorized as "CS-Related" were: Business Information Systems, Computer and Information Systems, Computer Network Security, Computer Science, Computer Security and Information Assurance, Cyber Defense and Information Assurance, Cyber Intel and Security, Cybersecurity, Information Science, IT, and Management Information Systems. Those categorized as "CEG-Related" were: Computer Engineering, Computer Systems Engineering, Electrical Engineering. Other majors were: Aerospace Engineering, Chemical Engineering, Drafting Technologies, Mathematics, and Political Science. In 2018, one student had as dual major in Computer Science and Computer Engineering and is counted in both sets.

S3) "This lesson should continue to be taught as part of ACE."

In 2019, they responded to the following questions.

S1) "How fun was the lab?" (from 1="Not fun at all" to 5="Very fun").
S2) "How much did this lab help me understand capturing and analyzing computer network traffic?" (from 1="Not helpful at all" to 5="Very helpful").
S3) "How much did this lab promote team communication?" (from 1= "No communication at all" to 5="Significant communication").
S4) "How much did this lab promote teamwork?" (from 1="No teamwork at all" to 5="Significant teamwork").
S5) "How much did this lab promote team problem solving?" (from 1="No problem solving at all" to 5="Significant problem solving").

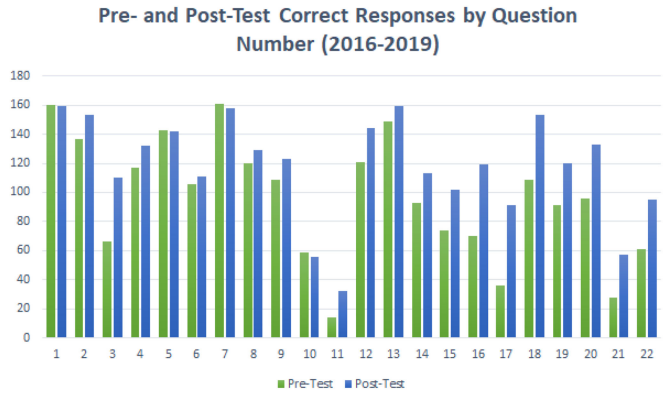
Finally, a brief group discussion took place about the gameplay experience.

B. Results

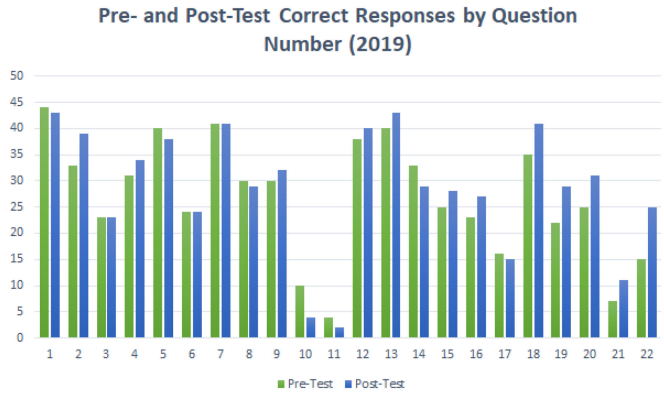
Fig. 5 presents participant demographic data. Fig. 6 indicates the number of students that answered each test question correctly on the pretest and the number that answered correctly on the posttest. Cumulative results for all four offerings are shown, as well as results for just the 2019 offering.

Fig. 7 depicts various order statistics of the overall pre- and posttest score sample distributions, for both the combined results of 2016–2018 and for 2019. No test scores qualified as outliers by falling 1.5 times the interquartile range below the first quartile or above the third quartile. Test scores for 2019 were tested against both the overall statistics and those for 2019 alone.

Wilcoxon signed rank tests are used to evaluate the hypotheses for the years 2016–2018 and 2019 that the pre and posttest scores are drawn from the same distribution. The test for the



(a) 2016–2019



(b) 2019

Fig. 6. (a) Total number of students correctly answering each question on the pretest and posttest over the four offerings (2016–2019). (b) Number of students in the 2019 offering answering correctly.

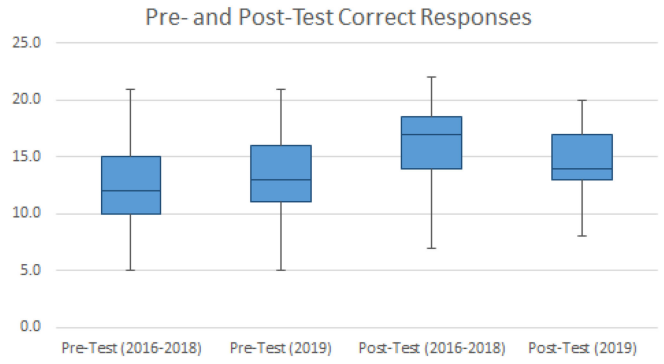


Fig. 7. Box and whisker plots for 2016–2018 and 2019 pre- and posttest score sample distributions. Each plot indicates the sample maximum, minimum, quartiles, and median. No test scores qualified as outliers.

2016–2018 scores yields $z = -8.5837$ ($p < 0.00001$) indicating that the posttest scores are significantly higher. The corresponding test for the 2019 scores yields $z = -3.3191$ ($p = 0.00045$) again, indicating that the posttest scores are significantly higher.

Similarly, Kruskal–Wallis tests are used to evaluate the hypotheses that the 2016–2018 pretest (posttest) scores and those from 2019 come from the same distribution. The test for the pretest scores yields $H = 1.6774$ ($p = 0.19527$), indicating that the two sets of scores are not significantly different. In contrast, the corresponding test for the posttest scores yields $H = 9.5694$ ($p = 0.00198$), indicating that the 2019 posttest scores are significantly lower than those from 2016–2018.

The sentiment questionnaire results are presented in Table V.

TABLE V
STUDENT LIKERT DATA FOR THE CSO LESSON

2016 Responses [1]					
Stem	1	2	3	4	5
S1	0	0	0	35%	65%
S2	0	0	9%	26%	65%
S3	3%	3%	3%	26%	65%

2017 Responses [1]					
Stem	1	2	3	4	5
S1	0	0	0	23%	77%
S2	0	0	0	61%	39%
S3	0	0	0	30%	70%

2019 Responses [2]					
Stem	1	2	3	4	5
Q1	0	0	8%	38%	54%
Q2	3%	5%	24%	26%	42%
Q3	0	3%	15%	28%	54%
Q4	0	5%	5%	34%	55%
Q5	0	3%	5%	38%	54%

C. Discussion

Pre/posttest questions 1–11 of the pre/posttest are multiple choice and primarily ask the student to define terms related to networking (IP, UDP, Wireshark, IP address, hostname, subnet, domain name, media access control address, monitor mode, and promiscuous mode). The game does not define any of these terms for the students. As shown in Fig. 6, with few exceptions, the pretest scores on these questions were relatively high.

Questions 12–22 are a mixture of multiple choice and short answer and assess the students' ability to analyze computer networking data and software. Specifically, questions 12–17 require the students to analyze a stream of bytes from a captured packet and interpret its contents, while questions 18–22 require the students to analyze a snippet of C++ networking code and predict its behavior. These questions also relate more directly to the game—students use Wireshark to capture similar packets and then interpret them, and students use a similar code snippet to craft and send their own packets. In years 2016–2018, the increase in the number of students correctly answering these questions was for the most part greater than the increase seen for the terminology questions, indicating at least short term retention of skills learned through the intervention.

Because all relevant p-values from the Wilcoxon signed rank tests are less than 0.0005, the test concludes that the overall improvements in test scores both for 2016–2018 and for 2019 are statistically significant at the 99.95% confidence level. This is strong evidence that the cadets learned as a result

[1]For 2016 and 27, the response scale ranged from 1 = Strongly Disagree to 5 = Strongly Agree.
[2]For 2019, the response scale varied by question.

of the overall intervention in 2016–2018 and as a result of simply playing the game in 2019.

In addition, the conclusion from the Kruskal–Wallis test that the posttest scores for 2019 were significantly lower than those for 2016-2018 is statistically significant at the 99.5% confidence level. This is strong evidence that administering the pretest after the prebriefing is correlated with less effective learning. Although we are not prepared to conclude that this is a causal relationship, it does suggest the possibility that administering the pretest before the prebriefing primes the students so that they retain more of the content.

On the sentiment questionnaire for years 2016 and 2017, over 90% of students in each year responded either "Agree" or "Strongly Agree" for all three questions (see Table V). This is evidence of relevance, engagement, and enjoyment. More direct evidence of enjoyment is provided by the fact that 92% of 2019 students responded either 4 or 5 to Q1. Similarly, for relevance and engagement, at least 92% of students responded either 4 or 5 to questions Q2–Q5.

Based on qualitative observations during gameplay, the greatest determining factor of team success (in terms of completing or winning the game) is the ability of team members to communicate with each other to solve a problem. Progressing in the game requires coordinating many tasks, each of which can have its own substantial learning curve. For example, the 2016 Red team, which both scored the highest and was the only team to complete all game objectives, communicated well with each other. In addition, individual team members were quick to figure out what to do and how to do it. This particularly displayed itself with hacking, which is not essential for completing the game, but increases ranking both by earning points and by hindering other team's ability to earn points. The 2016 Red team showed initiative and enthusiasm by performing multiple hacks early in the game. In contrast, the 2016 Yellow team did not communicate well with each other at all, and they were the only team that did not find at least one clue. The three other teams took longer to figure things out, but were able to communicate reasonably well, as evidenced both by observation during the game and by the fact that they earned two or three clues each. Similar observations hold true for the 2017–2019 teams as well.

Perhaps most importantly from the cadets' perspective, however, everyone has seemed to enjoy the game. The virtual world has a strong visual appeal; during setup, teams play around with their joystick controllers and have fun moving around and seeing the results on other teams' projectors. The game has been difficult enough to engage all cadets, but not difficult enough to make them give up; all teams have continued to work at the game until the very end of the time available, and only the 2016 Yellow team lost hope of success. The 2016 Red team enjoyed themselves so much that they took a group photo afterwards.

VII. Conclusion

CSO is intended to give students practical experience with computer networks in a fun and engaging way. Four years of results from the ACE program intervention indicate that it does so. During the lesson when the game has been played, the students all appear to be motivated and focused on succeeding, and most have commented afterward that they enjoyed playing. Beginning with 2019, we first gave the pregame lecture followed by a pretest, playing the game, and a posttest. This minor restructure ensured that results on the posttest did not include learning from the pregame lecture. However, there is also some reason to believe that this change reduced the overall effectiveness of the intervention. Additionally, in 2019, we also began to formally collect additional sentiment data to measure perception of related aspects of the game, such as teamwork, communication, problem solving, and enjoyment.

Furthermore, quantitative analysis of the data collected indicates that the intervention is effective as a teaching tool. Overall, there was a statistically significant increase in scores from the pretest to the posttest. In addition, a question-by-question analysis shows a greater improvement on the application level questions than on the knowledge and comprehension level questions, indicating an increase in practical knowledge (see Fig. 6).

Our current results do not assess the participants' transfer of learning to situations outside the CSO lesson. Doing so meaningfully within the three-week ACE program would be difficult at best. However, we intend to release the CSO game and all accompanying materials involved in the intervention for use by other educational institutions. Researchers adopting it for use in a more traditional setting would be able to assess the learning that persists over the duration of a full academic term.

References

[1] D. Crookall, "Serious games, debriefing, and simulation/gaming as a discipline," *Simul. Gaming*, vol. 41, no. 6, pp. 898–920, Jan. 2011.

[2] S. Tang and M. Hanneghan, "Fusing games technology and pedagogy for games-based learning through a model driven approach," in *Proc. IEEE Colloq. Humanities, Sci. Eng.*, Dec. 2011, pp. 380–385.

[3] C. Johnson *et al.*, "Game development for computer science education," in *Proc. ITiCSE Working Group Rep.*, Jul. 11–13, 2016, pp. 23–44.

[4] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer, *Serious Games: Foundations, Concepts and Practice*. Cham, Switzerland: Springer, 2016.

[5] J. Huizinga, *Homo Ludens: A Study of the Play-Element in Culture*. Boston, MA, USA: Beacon Press, 1955.

[6] A. C. Graesser and V. Ottati, "Why stories? Some evidence, questions, and challenges," in *Knowledge and Memory: The Real Story*, R. C. Wyer, Ed. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1995, vol. 8, ch. 5, pp. 121–132.

[7] M. M. McGill *et al.*, "If memory serves: Towards designing and evaluating a game for teaching pointers to undergraduate students," in *Proc. ITiCSE Conf. Working Group Rep.*, Jul. 2018, pp. 25–46.

[8] D. R. Michael and S. L. Chen, *Serious Games: Games That Educate, Train, and Inform*. Boston, MA, USA: Muska and Lipman/Premier-Trade, 2005.

[9] P. Petridis *et al.*, "State-of-the-art in business games," *Int. J. Serious Games*, vol. 2, no. 1, pp. 56–69, Feb. 2015.

[10] M. Hendrix, A. Al-Sherbaz, and V. Bloom, "Game based cyber security training: Are serious games suitable for cyber security training?" *Int. J. Serious Games*, vol. 3, no. 1, pp. 2384–8766, Mar. 2016.

[11] B. D. Cone, C. E. Irvine, M. F. Thompson, and T. D. Nguyen, "A video game for cyber security training and awareness," *Comput. Secur.*, vol. 26, pp. 63–72, Feb. 2007.

[12] X. S. Chang and K. Y. Chua, "A CyberCIEGE traffic analysis extension for teaching network security," M.S. thesis, Dept. Comput. Sci., Naval Postgraduate School, Monterey, CA, USA, 2011.

[13] *WireShark*, 3.2.2 release. [Online]. Available: https://www.wireshark.org, Accessed on Mar. 26, 2020.

[14] C. Jordan, M. Knapp, D. Mitchell, M. Claypool, and K. Fisler, "Countermeasures: A game for teaching computer security," in *Proc. 10th Annu. Workshop Netw. Syst. Support Games*, Oct. 2011, pp. 7:1–7:6.
[15] D. Twitchell, "SecurityCom: A multi-player game for researching and teaching information security teams," *J. Digit. Forensics, Secur. Law*, vol. 2, 2007, Art. no. 1.
[16] R. A. G. Gultom and B. Alrianto, "Enhancing network security environment by empowering modeling and simulation strategy," in *Proc. 11th Int. Conf. Internet Monit. Protection*, May 2016, pp. 45–52.
[17] I/ITSEC, "Serious Games Showcase and Challenge (SGS&C)," 2018. [Online]. Available: https://www.iitsec.org/education/serious-games, Accessed on Mar. 26, 2020.
[18] *HackNet*. Team Fractal Alligator. [Online]. Available: http://www.hacknet-os.com/, Accessed on Mar. 26, 2020.
[19] X. Yuan, P. Vega, Y. Qadah, R. Archer, H. Yu, and J. Xu, "Visualization tools for teaching computer security," *ACM Trans. Comput. Educ.*, vol. 9, no. 4, Jan. 2010, Art. no. 20.
[20] AETC/A3/AAD, "Air Force Handbook 1," 2017. [Online]. Available: https://static.e-publishing.af.mil/production/1/af_a1/publication/afhandbook1/afhandbook1.pdf
[21] Department of Defense, "Department of defense cyber strategy," 2018. [Online]. Available: https://archive.defense.gov/home/features/2015/0415_cyber-strategy/Final_2015_DoD_CYBER_STRATEGY_-for_web.pdf, Accessed on Mar. 26, 2020.
[22] The Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf, Accessed on Mar. 26, 2020.
[23] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed., Reading, MA, USA: Pearson, 2017.
[24] L. W. Anderson and D. R. Krathwohl, Eds., *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*, 2nd ed., New York, NY, USA: Allyn & Bacon, Dec. 2001.
[25] S. Oviatt, "Multimodal interfaces," in *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 2002, pp. 286–304.
[26] S. Nykl, "Cyber space odyssey: Cyber education and network forensics." [Online]. Available: https://www.youtube.com/watch?v=fn9QNoVsnZI, Accessed on Mar. 26, 2020.
[27] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *Proc. 19th Nat. Conf. Artif. Intell. Challenges Game AI Workshop,*, 2004, pp. 1–5.
[28] M. A. Scott, "A monument to the player: Preserving a landscape of socio-cultural capital in the transitional MMORPG," *New Rev. Hypermedia Multimedia*, vol. 18, no. 4, pp. 295–320, Dec. 2012.
[29] S. Nykl, C. Mourning, M. Leitch, D. Chelberg, T. Franklin, and C. Liu, "An overview of the STEAMiE educational game engine," in *Proc. 38th Annu. Frontiers Educ. Conf.*, Oct. 2008, pp. F3B-21–F3B-25.

Kendra Graham is currently working toward the bachelor's degree in computer science with Brigham Young University, Provo, UT, USA.
She specializes in computer animation and graphics.

James Anderson is currently working toward the bachelor's degree in computer science with Wright State University, Dayton, OH, USA.
His research interests include data visualization and 3D graphics.

Conrad Rife is currently working toward the bachelor's degree in computer science with Brigham Young University - Idaho, Rexburg, ID, USA.
His research interests include computer graphics and networking.

Bryce Heitmeyer is currently working toward the bachelor's degree in computer science with Wright State University, Dayton, OH, USA.
His research interests include computer graphics and machine learning.

Pranav R. Patel is currently working toward the Ph.D. degree in electrical engineering with the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA.
His research interests include RF and hardware digital design.

Scott Nykl received the B.S. degree in software engineer from UW-Platteville, Platteville, WI, USA, in 2006, and the M.S. and Ph.D. degrees in computer science from Ohio University, Athens, OH, USA, in 2008 and 2013, respectively.
He is currently an Associate Professor of Computer Science with the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. His research interests include 3-D computer graphics and computer vision.

Alan C. Lin received the B.S. degree in computer engineering from Rutgers University, New Brunswick, NJ, USA, in 2004, and the M.S. and Ph.D. degrees in computer science from the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, USA, in 2008 and 2015, respectively.
He is currently an International Program Officer with the Air Force Office of Scientific Research, Wright-Patterson Air Force Base. His research interests include autonomy, cybersecurity, and serious games.

Laurence D. Merkle received the B.S. degree in computers and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1987, and the M.S.C.E. and Ph.D. degrees in computer engineering from the Air Force Institute of Technology (AFIT), Wright-Patterson Air Force Base, OH, USA, in 1992 and 1996, respectively.
He is currently an Assistant Professor of Computer Science with the AFIT. His research interests include quantum computing, evolutionary computation, and computer science education.